

Python: module cdms.coord

cdms.coord

[index](#)

CDMS CoordinateAxis objects

Modules

[MA](#)

[Numeric](#)

[PropertiedClasses](#)

[cdms.cdmsNode](#)

[cdtime](#)

[copy](#)

[cdms.internattr](#)

[string](#)

[types](#)

Classes

[cdms.cdmsobj.CdmsObj\(cdms.internattr.InternalAttributesClass\)](#)

[AbstractCoordinateAxis](#)

[AbstractAxis2D](#)

[DatasetAxis2D\(AbstractAxis2D, cdms.variable.DatasetVariable\)](#)

[FileAxis2D\(AbstractAxis2D, cdms.fvariable.FileVariable\)](#)

[TransientAxis2D\(AbstractAxis2D, cdms.tvariable.TransientVariable\)](#)

class ***AbstractAxis2D***([AbstractCoordinateAxis](#))

Method resolution order:

[AbstractAxis2D](#)

[AbstractCoordinateAxis](#)

[cdms.cdmsobj.CdmsObj](#)

[cdms.internattr.InternalAttributesClass](#)

[PropertiedClasses.Properties.PropertiedClass](#)

Methods defined here:

__init__(self, parent=None, variableNode=None, bounds=None)

clone(self, copyData=1)

clone (self, copyData=1)

Return a copy of self as a transient axis.

If copyData is 1, make a separate copy of the data.

setBounds(self, bounds)

subSlice(self, *specs, **keys)

Methods inherited from [AbstractCoordinateAxis](#):

```
designateLatitude(self, persistent=0)
    # Designate axis as a latitude axis.
    # If persistent is true, write metadata to the container.

designateLevel(self, persistent=0)
    # Designate axis as a vertical level axis
    # If persistent is true, write metadata to the container.

designateLongitude(self, persistent=0)
    # Designate axis as a longitude axis
    # If persistent is true, write metadata to the container.

designateTime(self, persistent=0, calendar=4369)
    # Designate axis as a time axis, and optionally set the calendar.
    # If persistent is true, write metadata to the container.

getBounds(self)

getCalendar(self)
    # Return the cdtime calendar: GregorianCalendar, NoLeapCalendar,
    # or None. If the axis does not have a calendar attribute, return
    # calendar.

getData(self)

getExplicitBounds(self)
    # Return None if not explicitly defined

info(self, flag=None, device=None)
    Write info about axis; include dimension values and weights if applicable.

isAbstractCoordinate(self)

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

listall(self, all=None)
    Get list of info about this axis.

setCalendar(self, calendar, persistent=1)
    # Set the calendar
```

`size(self, axis=None)`
Number of elements in array, or in a particular axis.

`writeToFile(self, file)`

Data and other attributes inherited from AbstractCoordinateAxis:

`axis_count = 0`

Methods inherited from cdms.cdmsobj.CdmsObj:

`dump(self, path=None, format=1)`

`dump(self, path=None, format=1)`

Dump an XML representation of this object to a file.

'path' is the result file name, None for standard output.

'format'==1 iff the file is formatted with newlines for readability.

`matchPattern(self, pattern, attribute, tag)`

Match a pattern in a string-valued attribute. If attribute
search all string attributes. If tag is not None, it must m

`matchone(self, pattern, attrname)`

Return true iff the attribute with name attrname is a string
attribute which matches the compiled regular expression pattern
if attrname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is n

`searchPattern(self, pattern, attribute, tag)`

Search for a pattern in a string-valued attribute. If attribute
search all string attributes. If tag is not None, it must m

`searchPredicate(self, predicate, tag)`

Apply a truth-valued predicate. Return a list containing a
if the predicate is true and either tag is None or matches
If the predicate returns false, return an empty list

`searchone(self, pattern, attrname)`

Return true iff the attribute with name attrname is a string
attribute which contains the compiled regular expression pattern
if attrname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is n
a string.

Methods inherited from cdms.internattr.InternalAttributesClass:

`is_internal_attribute(self, name)`

`is_internal_attribute(name)` is true if name is internal.

`replace_external_attributes(self, newAttributes)`

`replace_external_attributes(newAttributes)`

Replace the external attributes with dictionary newAttributes

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

`delattr(self, name)`

`getattr(self, name)`

`setattr(self, name, value)`

`get_property_d(self, name)`

Return the 'del' property handler for name that self uses.
Returns None if no handler.

`get_property_g(self, name)`

Return the 'get' property handler for name that self uses.
Returns None if no handler.

`get_property_s(self, name)`

Return the 'set' property handler for name that self uses.
Returns None if no handler.

`set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)`

Set attribute handlers for name to methods actg, acts, actd
None means no change for that action.

nowrite = 1 prevents setting this attribute.

nowrite defaults to 0.

nodelete = 1 prevents deleting this attribute.

nodelete defaults to 1 unless actd given.

if nowrite and nodelete is None: nodelete = 1

class ***AbstractCoordinateAxis***([cdms.cdmsobj.CdmsObj](#))

AbstractCoordinateAxis defines the common interface
for coordinate variables/axes.

Method resolution order:

[AbstractCoordinateAxis](#)

[cdms.cdmsobj.CdmsObj](#)

[cdms.internalattr.InternalAttributesClass](#)

[PropertiedClasses.Properties.PropertiedClass](#)

Methods defined here:

`init(self, parent=None, variableNode=None, bounds=None)`

`clone(self, copyData=1)`

clone (self, copyData=1)

Return a copy of self as a transient axis.

If copyData is 1, make a separate copy of the data.

```

designateLatitude(self, persistent=0)
    # Designate axis as a latitude axis.
    # If persistent is true, write metadata to the container.

designateLevel(self, persistent=0)
    # Designate axis as a vertical level axis
    # If persistent is true, write metadata to the container.

designateLongitude(self, persistent=0)
    # Designate axis as a longitude axis
    # If persistent is true, write metadata to the container.

designateTime(self, persistent=0, calendar=4369)
    # Designate axis as a time axis, and optionally set the calendar
    # If persistent is true, write metadata to the container.

getBounds(self)

getCalendar(self)
    # Return the cdtime calendar: GregorianCalendar, NoLeapCalendar,
    # or None. If the axis does not have a calendar attribute, return
    # calendar.

getData(self)

getExplicitBounds(self)
    # Return None if not explicitly defined

info(self, flag=None, device=None)
    Write info about axis; include dimension values and weights if applicable.

isAbstractCoordinate(self)

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

listall(self, all=None)
    Get list of info about this axis.

setBounds(self, bounds)

setCalendar(self, calendar, persistent=1)
    # Set the calendar

```

`size(self, axis=None)`
Number of elements in array, or in a particular axis.

`writeToFile(self, file)`

Data and other attributes defined here:

`axis_count = 0`

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

`dump(self, path=None, format=1)`

`dump(self, path=None, format=1)`

Dump an XML representation of this object to a file.

'path' is the result file name, None for standard output.

'format'==1 iff the file is formatted with newlines for readability.

`matchPattern(self, pattern, attribute, tag)`

Match a pattern in a string-valued attribute. If attribute is None, search all string attributes. If tag is not None, it must match tag.

`matchone(self, pattern, attrname)`

Return true iff the attribute with name attrname is a string attribute which matches the compiled regular expression pattern. If attrname is None and pattern matches at least one string attribute. Return false if the attribute is not found or is not a string.

`searchPattern(self, pattern, attribute, tag)`

Search for a pattern in a string-valued attribute. If attribute is None, search all string attributes. If tag is not None, it must match tag.

`searchPredicate(self, predicate, tag)`

Apply a truth-valued predicate. Return a list containing a string if the predicate is true and either tag is None or matches tag. If the predicate returns false, return an empty list.

`searchone(self, pattern, attrname)`

Return true iff the attribute with name attrname is a string attribute which contains the compiled regular expression pattern. If attrname is None and pattern matches at least one string attribute. Return false if the attribute is not found or is not a string.

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

`is_internal_attribute(self, name)`

`is_internal_attribute(name)` is true if name is internal.

`replace_external_attributes(self, newAttributes)`

`replace_external_attributes(newAttributes)`

Replace the external attributes with dictionary newAttributes.

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

`__delattr__(self, name)`

`__getattr__(self, name)`

`__setattr__(self, name, value)`

`get_property_d(self, name)`

Return the 'del' property handler for name that self uses.
Returns None if no handler.

`get_property_g(self, name)`

Return the 'get' property handler for name that self uses.
Returns None if no handler.

`get_property_s(self, name)`

Return the 'set' property handler for name that self uses.
Returns None if no handler.

`set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)`

Set attribute handlers for name to methods actg, acts, actd
None means no change for that action.

nowrite = 1 prevents setting this attribute.

nowrite defaults to 0.

nodelete = 1 prevents deleting this attribute.

nodelete defaults to 1 unless actd given.

if nowrite and nodelete is None: nodelete = 1

class ***DatasetAxis2D***([AbstractAxis2D](#), [cdms.variable.DatasetVariable](#))

Two-dimensional coordinate axis in a dataset.

Method resolution order:

[DatasetAxis2D](#)

[AbstractAxis2D](#)

[AbstractCoordinateAxis](#)

[cdms.cdmsobj.CdmsObj](#)

[cdms.internattr.InternalAttributesClass](#)

[PropertiedClasses.Properties.PropertiedClass](#)

[cdms.variable.DatasetVariable](#)

[cdms.avariable.AbstractVariable](#)

[cdms.slabinterface.Slab](#)

Methods defined here:

`__init__(self, parent, id=None, variableNode=None, bounds=None)`

Note: node is a VariableNode

```
__repr__(self)
```

Methods inherited from AbstractAxis2D:

```
clone(self, copyData=1)
    clone (self, copyData=1)
        Return a copy of self as a transient axis.
        If copyData is 1, make a separate copy of the data.
```

```
setBounds(self, bounds)
```

```
subSlice(self, *specs, **keys)
```

Methods inherited from AbstractCoordinateAxis:

```
designateLatitude(self, persistent=0)
    # Designate axis as a latitude axis.
    # If persistent is true, write metadata to the container.
```

```
designateLevel(self, persistent=0)
    # Designate axis as a vertical level axis
    # If persistent is true, write metadata to the container.
```

```
designateLongitude(self, persistent=0)
    # Designate axis as a longitude axis
    # If persistent is true, write metadata to the container.
```

```
designateTime(self, persistent=0, calendar=4369)
    # Designate axis as a time axis, and optionally set the calendar.
    # If persistent is true, write metadata to the container.
```

```
getBounds(self)
```

```
getCalendar(self)
```

```
    # Return the cdtime calendar: GregorianCalendar, NoLeapCalendar,
    # or None. If the axis does not have a calendar attribute, return
    # calendar.
```

```
getData(self)
```

```
getExplicitBounds(self)
```

```
    # Return None if not explicitly defined
```

```
info(self, flag=None, device=None)
```

```
    Write info about axis; include dimension values and weights if
```

```
isAbstractCoordinate(self)
```

```
isLatitude(self)
```

```
    # Return true iff the axis is a latitude axis
```

```

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

listall(self, all=None)
    Get list of info about this axis.

setCalendar(self, calendar, persistent=1)
    # Set the calendar

size(self, axis=None)
    Number of elements in array, or in a particular axis.

writeToFile(self, file)

```

Data and other attributes inherited from [AbstractCoordinateAxis](#):

axis_count = 0

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

```

dump(self, path=None, format=1)
    dump(self, path=None, format=1)
    Dump an XML representation of this object to a file.
    'path' is the result file name, None for standard output.
    'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)
    # Match a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

matchone(self, pattern, attrname)
    # Return true iff the attribute with name attrname is a string
    # attribute which matches the compiled regular expression pattern.
    # if attrname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

```

***searchone*(self, pattern, atname)**

Return true iff the attribute with name atname is a string attribute which contains the compiled regular expression pattern if atname is None and pattern matches at least one string attribute. Return false if the attribute is not found or is not a string.

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

***is_internal_attribute*(self, name)**

is internal attribute(name) is true if name is internal.

***replace_external_attributes*(self, newAttributes)**

replace external attributes(newAttributes)

Replace the external attributes with dictionary newAttributes

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

***get_property_d*(self, name)**

Return the 'del' property handler for name that self uses.

Returns None if no handler.

***get_property_g*(self, name)**

Return the 'get' property handler for name that self uses.

Returns None if no handler.

***get_property_s*(self, name)**

Return the 'set' property handler for name that self uses.

Returns None if no handler.

***set_property*(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)**

Set attribute handlers for name to methods actg, acts, actd

None means no change for that action.

nowrite = 1 prevents setting this attribute.

nowrite defaults to 0.

nodelete = 1 prevents deleting this attribute.

nodelete defaults to 1 unless actd given.

if nowrite and nodelete is None: nodelete = 1

Methods inherited from [cdms.variable.DatasetVariable](#):

__getitem__(self, key)

__getslice__(self, low, high)

```

len(self)
    Length of first dimension

setitem(self, index, value)

selslice(self, low, high, value)

expertPaths(self, slist)
    expertPaths(self, slicelist)
        takes a list of slices,
        returns a 3-tuple: (npart, dimensionlist, partitionSlices) where
        npart is the number of partitioned dimensions: 0, 1, or 2;
        dimensionlist is a tuple of length npart, having the dimension
        numbers of the partitioned dimensions;
        partitionSlices is the list of file-specific (filename, slice)
        corresponding to the paths and slices within the files to be
        The exact form of partitionSlices depends on the value of npart
        npart      partitionSlices
        0          (filename,slicelist)
        1          [(filename,slicelist),..., (filename,slicelist)]
        2          [[[filename,slicelist],..., (filename,slicelist)]]
                      [(filename,slicelist),..., (filename,slicelist)]
                      ...
                      [(filename,slicelist),..., (filename,slicelist)]]

```

Note:

- A filename of None indicates that no file was found with corresponding to the slicelist.
- If partitionSlices is None, the slicelist does not intersect.
- An empty partitionSlices [] means that the variable is zero.

expertSlice(self, initlist)

genMatch(self, axis, interval, matchnames)

Helper function for expertPaths.
axis is a partitioned axis, either time or vertical level.
interval is an index interval (istart, iend).
matchnames is a partially filled list [id, timestamp, timeend].
If a filemap is used, matchnames has indices, otherwise has strings.

Function modifies matchnames based on axis and interval,
returns the modified matchnames tuple.

getAxis(self, n)

getDomain(self)

getFilepath(self, matchnames, template)

Lookup or generate the file path, depending on whether a file
or template is present.

getPaths(self, *specs, **keys)

```

# Get the paths associated with the interval region specified
# by 'intervals'. This incorporates most of the logic of __ge__
# without actually reading the data.
#
# 'specs' is a list of interval range specifications as defined
# for getSlice.
#
# The function returns a list of tuples of the form (path,slicetuple)
# where path is the path of a file, and slicetuple is a tuple
# of slices, of the same length as the rank of the variable, representing
# region of the variable which is contained in the file. The
# would retrieve the data for that file:
#
#     f = Cdunif.CdunifFile(path,'r')
#     var = f.variables[self.name_in_file]
#     data = apply(var.getitem,slicelist)

getShape(self)

getTemplate(self)
    # Get the template

getValue(self, squeeze=1)
    Return the entire set of values.

initDomain(self, axisdict, griddict)
    Must be called by whoever made this Variable to set up axes.

typecode(self)

```

Methods inherited from [cdms.avariable.AbstractVariable](#):

```

__abs__(self)

__add__(self, other)

__array__(self, t=None)

__call__(self, *args, **kwargs)
    Selection of a subregion using selectors

__div__(self, other)

__eq__(self, other)

__ge__(self, other)

__gt__(self, other)

__iadd__(self, other)
    Add other to self in place.

```

`__idiv__(self, other)`
 Divide self by other in place.

`__imul__(self, other)`
 Multiply self by other in place.

`__isub__(self, other)`
 Subtract other from self in place.

`__le__(self, other)`

`__lshift__(self, n)`

`__lt__(self, other)`

`__mul__(self, other)`

`__ne__(self, other)`

`__neg__(self)`

`__pow__(self, other, third=None)`

`__radd__ = __add__(self, other)`

`__rdiv__(self, other)`

`__rmul__ = __mul__(self, other)`

`__rshift__(self, n)`

`__rsub__(self, other)`

`__sqrt__(self)`

`__sub__(self, other)`

`assignValue(self, data)`

`astype(self, tc)`
 return self as array of given type.

`crossSectionRegrid(self, newLevel, newLatitude, missing=None, order=None, method='log')`
 Return the variable regridded to new pressure levels and latitudes.
 The variable should be a function of lat, level, and (optional) time.
 <newLevel> is an axis of the result pressure levels.
 <newLatitude> is an axis of latitude values.
 <method> is optional, either "log" to interpolate in the log
 or "linear" for linear interpolation.
 <missing> and <order> are as for regrid.CrossSectionRegridder.

`decode(self, ar)`

Decode compressed data. ar is a masked array, scalar, or MA.

***generateGridkey*(self, convention, vardict)**
generateGridkey(): Determine if the variable is gridded, and generate ((latname, lonname, order, maskname, class), lat or (None, None, None) if not gridded. vardict is the variable

***generateRectGridkey*(self, lat, lon)**
generateRectGridkey(): Determine if the variable is gridded, and generate (latname, lonname, order, maskname, class) if gridded or None if not gridded

***getAxisIds*(self)**
Get a list of axis identifiers

***getAxisIndex*(self, axis_spec)**
Return the index of the axis specified by axis_spec.
Argument axis_spec and be as for axisMatches
Return -1 if no match.

***getAxisList*(self, axes=None, omit=None, order=None)**
Get the list of axis objects;
If axes is not None, include only certain axes.
If omit is not None, omit those specified by omit.
Arguments omit or axes may be as specified in axisMatchAxis
order is an optional string determining the output order

***getAxisListIndex*(self, axes=None, omit=None, order=None)**
Return a list of indices of axis objects;
If axes is not None, include only certain axes.
less the ones specified in omit. If axes is None,
use all axes of this variable.
Other specifications are as for axisMatchIndex.

***getConvention*(self)**
Get the metadata convention associated with this object.

***getGrid*(self)**
Return the grid

***getGridIndices*(self)**
Return a tuple of indices corresponding to the variable grid.

***getLatitude*(self)**
Get the first latitude dimension, or None if not found.

***getLevel*(self)**
Get the first vertical level dimension in the domain,
or None if not found.

***getLongitude*(self)**
Get the first longitude dimension, or None if not found.

`getMissing(self, asarray=0)`

Return the missing value as a scalar, or as a Numeric array if asarray==1

`getOrder(self, ids=0)`

getOrder(ids=0) returns the order string, such as tzyx.

if ids == 0 (the default) for an axis that is not t,z,x,y the order string will contain a '-' in that location. The result string will be of the same length as the number of axes. This makes it easy to loop over the dimensions.

if ids == 1 those axes will be represented in the order string as (id) where id is that axis' id. The result will be suitable for passing to order2index to get the corresponding axes, and to orderparse for dividing up into components.

`getRegion(self, *specs, **keys)`

getRegion

Read a region of data. A region is an n-dimensional rectangular region specified in coordinate space.

'slices' is an argument list, each item of which has one of the following forms:

- `x`, where `x` is a scalar
Map the scalar to the index of the closest coordinate value.
- `(x, y)`
Map the half-open coordinate interval $[x, y)$ to index interval.
- `(x, y, 'cc')`
Map the closed interval $[x, y]$ to index interval. Other options are 'oc' (open on the left), and 'co' (open on the right, the default).
- `(x, y, 'co', cycle)`
Map the coordinate interval with wraparound. If no cycle is specified, it will occur iff `axis.isCircular()` is true.
NOTE: Only one dimension may be wrapped.
- Ellipsis
Represents the full range of all dimensions bracketed by None.
- `:` or None
Represents the full range of one dimension.

For example, suppose the variable domain is `(time,level,lat,lon)`.

getRegion((10,20),850,Ellipsis,(-180,180))

retrieves:

- all times `t` such that $10 \leq t < 20$.
- level 850
- all values of all dimensions between level and lon (namely time, lat, lon).
- longitudes `x` such that $-180 \leq x < 180$. This will be wrapped if `lon.topology=='linear'`

`getSlice(self, *specs, **keys)`

`x.getSlice` takes arguments of the following forms and produce a return array. The keyword argument `squeeze` determines whether or not the shape of the returned array contains dimensions whose length is 1; by default this argument is 1, and such dimensions are 'squeezed out'.

There can be zero or more positional arguments, each of the form

(a) a single integer `n`, meaning `slice(n, n+1)`

(b) an instance of the slice class

(c) a tuple, which will be used as arguments to create a slice

(d) `None` or `:`, which means a slice covering that entire dimension

(e) Ellipsis `(...)`, which means to fill the slice list with 'Ellipsis' leaving only enough room at the end for the remaining positional arguments

There can be keyword arguments of the form `key = value`, where `key` can be one of the names '`time`', '`level`', '`latitude`', or '`longitude`'. The corresponding value can be any of (a)-(d) above.

There must be no conflict between the positional arguments and the keywords.

In (a)-(c) negative numbers are treated as offsets from the end of that dimension, as in normal Python indexing.

`getTime(self)`

Get the first time dimension, or `None` if not found

`isEncoded(self)`

True iff `self` is represented as packed data.

`pressureRegrid(self, newLevel, missing=None, order=None, method='log')`

Return the variable regridded to new pressure levels.

The variable should be a function of `lat`, `lon`, `pressure`, and `<newLevel>` is an axis of the result pressure levels.
`<method>` is optional, either "`log`" to interpolate in the log or "`linear`" for linear interpolation.
`<missing>` and `<order>` are as for `regrid.PressureRegridder`.

`rank(self)`

`reg_specs2slices(self, initSpecList, force=None)`

`regrid(self, togrid, missing=None, order=None, mask=None)`

return self regridded to the new grid. Keyword arguments are as for `regrid.Regridder`.

`reorder(self, order)`

return self reordered per the specification order

`select = __call__(self, *args, **kwargs)`

Selection of a subregion using selectors

`setGrid(self, grid)`

```
# Set the variable grid

setMissing(self, value)
    Set the missing value, which may be a scalar,
    a single-valued Numeric array, or None. The value is
    cast to the same type as the variable.

specs2slices(self, speclist, force=None)
    Create an equivalent list of slices from an index specification.
    An index specification is a list of acceptable items, which are
    -- an integer
    -- a slice instance (slice(start, stop, stride))
    -- the object "unspecified"
    -- the object None
    -- a colon
    The size of the speclist must be rank()

subRegion(self, *specs, **keys)
```

Methods inherited from cdms.slabinterface.Slab:

```
createattribute(self, name, value)
    Create an attribute and set its name to value.

deleteattribute(self, name)
    Delete the named attribute.

getattribute(self, name)
    Get the attribute name.

getdimattribute(self, dim, field)
    Get the attribute named field from the dim'th dimension.
    For bounds returns the old cu one-dimensional version.

listattributes(self)
    Return a list of attribute names.

listdimattributes(self, dim)
    List the legal axis field names.

listdimnames(self)
    Return a list of the names of the dimensions.

setattribute(self, name, value)
    Set the attribute name to value.

showdim(self)
    Show the dimension attributes and values.
```

Data and other attributes inherited from cdms.slabinterface.Slab:

```
std_slab_atts = ['filename', 'missing_value', 'comments', 'grid_name', 'grid_type', 'time_statistic', 'lo
```

```
class FileAxis2D(AbstractAxis2D, cdms.fvariable.FileVariable)
    # Two-dimensional coordinate axis in a file.
```

Method resolution order:

```
FileAxis2D
AbstractAxis2D
AbstractCoordinateAxis
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass
PropertiedClasses.Properties.PropertiedClass
cdms.fvariable.FileVariable
cdms.variable.DatasetVariable
cdms.avariable.AbstractVariable
cdms.slabinterface.Slab
```

Methods defined here:

```
__init__(self, parent, id, obj=None, bounds=None)

__repr__(self)
```

Methods inherited from **AbstractAxis2D**:

```
clone(self, copyData=1)
    clone (self, copyData=1)
    Return a copy of self as a transient axis.
    If copyData is 1, make a separate copy of the data.

setBounds(self, bounds)

subSlice(self, *specs, **keys)

Methods inherited from AbstractCoordinateAxis:

designateLatitude(self, persistent=0)
    # Designate axis as a latitude axis.
    # If persistent is true, write metadata to the container.

designateLevel(self, persistent=0)
    # Designate axis as a vertical level axis
    # If persistent is true, write metadata to the container.

designateLongitude(self, persistent=0)
    # Designate axis as a longitude axis
    # If persistent is true, write metadata to the container.
```

```

designateTime(self, persistent=0, calendar=4369)
    # Designate axis as a time axis, and optionally set the calendar.
    # If persistent is true, write metadata to the container.

getBounds(self)

getCalendar(self)
    # Return the cdtime calendar: GregorianCalendar, NoLeapCalendar,
    # or None. If the axis does not have a calendar attribute, return
    # calendar.

getData(self)

getExplicitBounds(self)
    # Return None if not explicitly defined

info(self, flag=None, device=None)
    Write info about axis; include dimension values and weights if applicable.

isAbstractCoordinate(self)

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

listall(self, all=None)
    Get list of info about this axis.

setCalendar(self, calendar, persistent=1)
    # Set the calendar

size(self, axis=None)
    Number of elements in array, or in a particular axis.

writeToFile(self, file)

```

Data and other attributes inherited from [AbstractCoordinateAxis](#):

axis_count = 0

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

dump(self, path=None, format=1)

```

dump(self, path=None, format=1)
    Dump an XML representation of this object to a file.
    'path' is the result file name, None for standard output.
    'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)
    # Match a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

matchone(self, pattern, attrname)
    # Return true iff the attribute with name attrname is a string
    # attribute which matches the compiled regular expression pattern
    # if attrname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a string
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, attrname)
    Return true iff the attribute with name attrname is a string
    attribute which contains the compiled regular expression pattern
    if attrname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not a string.

```

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

```

```
get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1
```

Methods inherited from [cdms.fvariable.FileVariable](#):

```
__len__(self)
    Length of first dimension.

__setitem__(self, index, value)

__setslice__(self, low, high, value)

assignValue(self, data)

expertSlice(self, initslicelist)

getValue(self, squeeze=1)
    Return the entire set of values.
```

```
initDomain(self, axisdict)
    Called by whoever made me.
```

```
typecode(self)
```

Methods inherited from [cdms.variable.DatasetVariable](#):

```
__getitem__(self, key)

__getslice__(self, low, high)

expertPaths(self, slist)
    expertPaths(self, slicelist)
    takes a list of slices,
    returns a 3-tuple: (npart, dimensionlist, partitionSlices) where
    npart is the number of partitioned dimensions: 0, 1, or 2;
    dimensionlist is a tuple of length npart, having the dimension
    numbers of the partitioned dimensions;
```

```

partitionSlices is the list of file-specific (filename, slice)
corresponding to the paths and slices within the files to be
The exact form of partitionSlices depends on the value of npart
npart      partitionSlices
0          (filename,slicelist)
1          [(filename,slicelist),..., (filename,slicelist)]
2          [[[filename,slicelist],..., (filename,slicelist)]]
            [(filename,slicelist),..., (filename,slicelist)]
...
[(filename,slicelist),..., (filename,slicelist)]]

```

Note:

- A filename of None indicates that no file was found with corresponding to the slicelist.
- If partitionSlices is None, the slicelist does not intersect.
- An empty partitionSlices [] means that the variable is zero-dimensional.

genMatch(self, axis, interval, matchnames)

Helper function for expertPaths.

axis is a partitioned axis, either time or vertical level.

interval is an index interval (istart, iend).

matchnames is a partially filled list [id, timestamp, timeend].

If a filemap is used, matchnames has indices, otherwise has strings.

Function modifies matchnames based on axis and interval, returns the modified matchnames tuple.

getAxis(self, n)

getDomain(self)

getFilePath(self, matchnames, template)

Lookup or generate the file path, depending on whether a file or template is present.

getPaths(self, *specs, **keys)

```

# Get the paths associated with the interval region specified
# by 'intervals'. This incorporates most of the logic of __getPaths()
# without actually reading the data.
#
# 'specs' is a list of interval range specifications as defined
# for getSlice.
#
# The function returns a list of tuples of the form (path,slicetuple)
# where path is the path of a file, and slicetuple is a tuple of
# slices, of the same length as the rank of the variable, representing
# region of the variable which is contained in the file. The
# would retrieve the data for that file.
#
#     f = Cdunif.CdunifFile(path,'r')
#     var = f.variables[self.name_in_file]
#     data = apply(vargetitem,slicelist)

```

```
getShape(self)

getTemplate(self)
    # Get the template
```

Methods inherited from [cdms.avariable.AbstractVariable](#):

abs (self)

add (self, other)

array (self, t=None)

call (self, *args, **kwargs)
Selection of a subregion using selectors

div (self, other)

eq (self, other)

ge (self, other)

gt (self, other)

iadd (self, other)
Add other to self in place.

idiv (self, other)
Divide self by other in place.

imul (self, other)
Multiply self by other in place.

isub (self, other)
Subtract other from self in place.

le (self, other)

lshift (self, n)

lt (self, other)

mul (self, other)

ne (self, other)

neg (self)

pow (self, other, third=None)

radd = ***add*** (self, other)

```

__rdiv__(self, other)
__rmul__ = __mul__(self, other)

__rshift__(self, n)

__rsub__(self, other)

__sqrt__(self)

__sub__(self, other)

astype(self, tc)
    return self as array of given type.

crossSectionRegrid(self, newLevel, newLatitude, missing=None, order=None, method='log')
    Return the variable regridded to new pressure levels and latitudes.
    The variable should be a function of lat, level, and (optional) time.
    <newLevel> is an axis of the result pressure levels.
    <newLatitude> is an axis of latitude values.
    <method> is optional, either "log" to interpolate in the log
        or "linear" for linear interpolation.
    <missing> and <order> are as for regrid.CrossSectionRegridder.

decode(self, ar)
    Decode compressed data. ar is a masked array, scalar, or MA masked array.

generateGridkey(self, convention, vardict)
    generateGridkey(): Determine if the variable is gridded,
    and generate ((latname, lonname, order, maskname, class), latname)
    or (None, None, None) if not gridded. vardict is the variable dictionary.

generateRectGridkey(self, lat, lon)
    generateRectGridkey(): Determine if the variable is gridded,
    and generate (latname, lonname, order, maskname, class) if gridded
    or None if not gridded

getAxisIds(self)
    Get a list of axis identifiers

getAxisIndex(self, axis_spec)
    Return the index of the axis specified by axis_spec.
    Argument axis_spec and be as for axisMatches
    Return -1 if no match.

getAxisList(self, axes=None, omit=None, order=None)
    Get the list of axis objects;
    If axes is not None, include only certain axes.
    If omit is not None, omit those specified by omit.
    Arguments omit or axes may be as specified in axisMatchAxis
    order is an optional string determining the output order

```

```

getAxisListIndex(self, axes=None, omit=None, order=None)
    Return a list of indices of axis objects;
    If axes is not None, include only certain axes.
    less the ones specified in omit. If axes is None,
    use all axes of this variable.
    Other specifiications are as for axisMatchIndex.

getConvention(self)
    Get the metadata convention associated with this object.

getGrid(self)
    # Return the grid

getGridIndices(self)
    Return a tuple of indices corresponding to the variable grid.

getLatitude(self)
    Get the first latitude dimension, or None if not found.

getLevel(self)
    Get the first vertical level dimension in the domain,
    or None if not found.

getLongitude(self)
    Get the first longitude dimension, or None if not found.

getMissing(self, asarray=0)
    Return the missing value as a scalar, or as
    a Numeric array if asarray==1

getOrder(self, ids=0)
    getOrder(ids=0) returns the order string, such as tzyx.

    if ids == 0 (the default) for an axis that is not t,z,x,y
    the order string will contain a '-' in that location.
    The result string will be of the same length as the number
    of axes. This makes it easy to loop over the dimensions.

    if ids == 1 those axes will be represented in the order
    string as (id) where id is that axis' id. The result will
    be suitable for passing to order2index to get the
    corresponding axes, and to orderparse for dividing up into
    components.

getRegion(self, *specs, **keys)
    getRegion
    Read a region of data. A region is an n-dimensional
    rectangular region specified in coordinate space.
    'slices' is an argument list, each item of which has one of the
    - x, where x is a scalar
        Map the scalar to the index of the closest coordinate value
    - (x, y)

```

Map the half-open coordinate interval [x,y) to index interval [x,y)

- (x,y,'cc')
Map the closed interval [x,y] to index interval. Other options are 'oc' (open on the left), and 'co' (open on the right, the case of 'co' is not yet implemented).
- (x,y,'co',cycle)
Map the coordinate interval with wraparound. If no cycle is specified, it will occur iff axis.isCircular() is true.
NOTE: Only one dimension may be wrapped.
- Ellipsis
Represents the full range of all dimensions bracketed by None.
- ':' or None
Represents the full range of one dimension.

For example, suppose the variable domain is (time,level,lat,lon)

```
getRegion((10,20),850,Ellipsis,(-180,180))
```

retrieves:

- all times t such that 10.≤t<20.
- level 850
- all values of all dimensions between level and lon (namely latitude and longitude)
- longitudes x such that -180≤x≤180. This will be wrapped around if lon.topology=='linear'

getSlice(self, *specs, **keys)

x.getSlice takes arguments of the following forms and produces a return array. The keyword argument squeeze determines whether or not the shape of the returned array contains dimensions whose length is 1; by default this argument is 1, and such dimensions are 'squeezed out'.

There can be zero or more positional arguments, each of the following forms:

- (a) a single integer n, meaning slice(n, n+1)
- (b) an instance of the slice class
- (c) a tuple, which will be used as arguments to create a slice object
- (d) None or ':', which means a slice covering that entire dimension
- (e) Ellipsis (...), which means to fill the slice list with 'Ellipsis' leaving only enough room at the end for the remaining positional arguments

There can be keyword arguments of the form key = value, where key can be one of the names 'time', 'level', 'latitude', or 'longitude'. The corresponding value can be any of (a)-(d) above.

There must be no conflict between the positional arguments and the keywords.

In (a)-(c) negative numbers are treated as offsets from the end of that dimension, as in normal Python indexing.

getTime(self)

Get the first time dimension, or None if not found

isEncoded(self)

True iff self is represented as packed data.

pressureRegrid(self, newLevel, missing=None, order=None, method='log')
 Return the variable regridded to new pressure levels.
 The variable should be a function of lat, lon, pressure, and
 <newLevel> is an axis of the result pressure levels.
 <method> is optional, either "log" to interpolate in the log
 or "linear" for linear interpolation.
 <missing> and <order> are as for regrid.PressureRegridder.

rank(self)

reg_specs2slices(self, initSpecList, force=None)

regrid(self, togrid, missing=None, order=None, mask=None)
 return self regridded to the new grid. Keyword arguments
 are as for regrid.Regridder.

reorder(self, order)
 return self reordered per the specification order

select = __call__(self, *args, **kwargs)
 Selection of a subregion using selectors

setGrid(self, grid)
 # Set the variable grid

setMissing(self, value)
 Set the missing value, which may be a scalar,
 a single-valued Numeric array, or None. The value is
 cast to the same type as the variable.

specs2slices(self, specList, force=None)
 Create an equivalent list of slices from an index specification.
 An index specification is a list of acceptable items, which are:
 -- an integer
 -- a slice instance (slice(start, stop, stride))
 -- the object "unspecified"
 -- the object None
 -- a colon
 The size of the specList must be *rank()*

subRegion(self, *specs, **keys)

Methods inherited from [cdms.slabinterface.Slab](#):

createattribute(self, name, value)
 Create an attribute and set its name to value.

deleteattribute(self, name)
 Delete the named attribute.

getattribute(self, name)
Get the attribute name.

getdimattribute(self, dim, field)
Get the attribute named field from the dim'th dimension.
For bounds returns the old cu one-dimensional version.

listattributes(self)
Return a list of attribute names.

listdimattributes(self, dim)
List the legal axis field names.

listdimnames(self)
Return a list of the names of the dimensions.

setattribute(self, name, value)
Set the attribute name to value.

showdim(self)
Show the dimension attributes and values.

Data and other attributes inherited from [cdms.slabinterface.Slab](#):

std_slab_atts = ['filename', 'missing_value', 'comments', 'grid_name', 'grid_type', 'time_statistic', 'lo

class ***TransientAxis2D***([AbstractAxis2D](#), [cdms.tvariable.TransientVariable](#))

Method resolution order:

[TransientAxis2D](#)
[AbstractAxis2D](#)
[AbstractCoordinateAxis](#)
[cdms.tvariable.TransientVariable](#)
[cdms.avariable.AbstractVariable](#)
[cdms.cdmsobj.CdmsObj](#)
[cdms.internattr.InternalAttributesClass](#)
[PropertiedClasses.Properties.PropertiedClass](#)
[cdms.slabinterface.Slab](#)
[MA.MA.MaskedArray](#)
[builtin .object](#)

Methods defined here:

__init__(self, data, typecode=None, copy=0, savespace=0, mask=None, fill_value=None, axes=None, copyaxes=1, bounds=None)

Create a transient 2D axis.

All arguments are as for [TransientVariable](#).

'bounds' is the bounds array, having shape (m,n,nvert) where
nvert is the max number of vertices per cell.

Methods inherited from [AbstractAxis2D](#):

clone(self, copyData=1)
 clone (self, copyData=1)
 Return a copy of self as a transient axis.
 If copyData is 1, make a separate copy of the data.

setBounds(self, bounds)

subSlice(self, *specs, **keys)

Methods inherited from [AbstractCoordinateAxis](#):

designateLatitude(self, persistent=0)
 # Designate axis as a latitude axis.
 # If persistent is true, write metadata to the container.

designateLevel(self, persistent=0)
 # Designate axis as a vertical level axis
 # If persistent is true, write metadata to the container.

designateLongitude(self, persistent=0)
 # Designate axis as a longitude axis
 # If persistent is true, write metadata to the container.

designateTime(self, persistent=0, calendar=4369)
 # Designate axis as a time axis, and optionally set the calendar
 # If persistent is true, write metadata to the container.

getBounds(self)

getCalendar(self)

 # Return the cdtime calendar: GregorianCalendar, NoLeapCalendar
 # or None. If the axis does not have a calendar attribute, return
 # calendar.

getData(self)

getExplicitBounds(self)

 # Return None if not explicitly defined

info(self, flag=None, device=None)

 Write info about axis; include dimension values and weights if applicable

isAbstractCoordinate(self)

isLatitude(self)

 # Return true iff the axis is a latitude axis

isLevel(self)

 # Return true iff the axis is a level axis

```

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

listall(self, all=None)
    Get list of info about this axis.

setCalendar(self, calendar, persistent=1)
    # Set the calendar

size(self, axis=None)
    Number of elements in array, or in a particular axis.

writeToFile(self, file)

```

Data and other attributes inherited from AbstractCoordinateAxis:

axis_count = 0

Methods inherited from cdms.tvariable.TransientVariable:

```

__len__(self)
    Length of first dimension

__repr__(self)

__str__(self)

assignValue(self, data)

astype(self, tc)
    return self as array of given type.

copyAxis(self, n, axis)
    Set n axis of self to a copy of axis. (0-based index)
    Invalidates grid.

copyDomain(self, other)
    Set the axes and grid by copying variable other.

copydimension(self, idim, other, jdim)
    Set idim dimension of self to variable other's jdim'th
    This is for old cu compatibility. Use copyAxis for new code.

expertSlice(self, slicelist)

getAxis(self, n)

getDomain(self)

```

```
getGrid(self)

getValue(self, squeeze=1)

initDomain(self, axes, copyaxes=1)

isEncoded(self)
    Transient variables are not encoded

setAxis(self, n, axis, savegrid=0)
    Set n axis of self to a copy of axis. (0-based index)

setAxisList(self, axislist)
    Set the axes to axislist.

setMaskFromGridMask(self, mask, gridindices)
    Set the mask for self, given a grid mask and the variable domain
    indices corresponding to the grid dimensions.

setMissing(self, value)
    Set missing value attribute and fill value

set_fill_value(self, value)
    Set missing value attribute and fill value

setdimattribute(self, dim, field, value)
    Set the attribute named field from the dim'th dimension.

typecode(self)
```

Data and other attributes inherited from [cdms.tvariable.TransientVariable](#):

count = 0

Methods inherited from [cdms.avariable.AbstractVariable](#):

abs_(self)

add_(self, other)

array_(self, t=None)

call_(self, *args, **kwargs)

Selection of a subregion using selectors

div_(self, other)

eq_(self, other)

ge_(self, other)

```

__getitem__(self, key)

__getslice__(self, low, high)

__gt__(self, other)

__iadd__(self, other)
    Add other to self in place.

__idiv__(self, other)
    Divide self by other in place.

__imul__(self, other)
    Multiply self by other in place.

__isub__(self, other)
    Subtract other from self in place.

__le__(self, other)

__lshift__(self, n)

__lt__(self, other)

__mul__(self, other)

__ne__(self, other)

__neg__(self)

__pow__(self, other, third=None)

__radd__ = __add__(self, other)

__rdiv__(self, other)

__rmul__ = __mul__(self, other)

__rshift__(self, n)

__rsub__(self, other)

__sqrt__(self)

__sub__(self, other)

crossSectionRegrid(self, newLevel, newLatitude, missing=None, order=None, method='log')
    Return the variable regridded to new pressure levels and latitudes.
    The variable should be a function of lat, level, and (optional) time.
    <newLevel> is an axis of the result pressure levels.
    <newLatitude> is an axis of latitude values.
    <method> is optional, either "log" to interpolate in the log

```

```
        or "linear" for linear interpolation.  
<missing> and <order> are as for regrid.CrossSectionRegridder  
  
decode(self, ar)  
    Decode compressed data. ar is a masked array, scalar, or MA.m  
  
generateGridkey(self, convention, vardict)  
    generateGridkey(): Determine if the variable is gridded,  
    and generate ((latname, lonname, order, maskname, class), lat  
    or (None, None, None) if not gridded. vardict is the variable  
  
generateRectGridkey(self, lat, lon)  
    generateRectGridkey(): Determine if the variable is gridded,  
    and generate (latname, lonname, order, maskname, class) if gr  
    or None if not gridded  
  
getAxisIds(self)  
    Get a list of axis identifiers  
  
getAxisIndex(self, axis_spec)  
    Return the index of the axis specified by axis_spec.  
    Argument axis_spec and be as for axisMatches  
    Return -1 if no match.  
  
getAxisList(self, axes=None, omit=None, order=None)  
    Get the list of axis objects;  
    If axes is not None, include only certain axes.  
    If omit is not None, omit those specified by omit.  
    Arguments omit or axes may be as specified in axisMatchAxis  
    order is an optional string determining the output order  
  
getAxisListIndex(self, axes=None, omit=None, order=None)  
    Return a list of indices of axis objects;  
    If axes is not None, include only certain axes.  
    less the ones specified in omit. If axes is None,  
    use all axes of this variable.  
    Other specifications are as for axisMatchIndex.  
  
getConvention(self)  
    Get the metadata convention associated with this object.  
  
getGridIndices(self)  
    Return a tuple of indices corresponding to the variable grid.  
  
getLatitude(self)  
    Get the first latitude dimension, or None if not found.  
  
getLevel(self)  
    Get the first vertical level dimension in the domain,  
    or None if not found.  
  
getLongitude(self)
```

Get the first latitude dimension, or None if not found.

`getMissing(self, asarray=0)`

Return the missing value as a scalar, or as a Numeric array if asarray==1

`getOrder(self, ids=0)`

`getOrder`(ids=0) returns the order string, such as tzyx.

if ids == 0 (the default) for an axis that is not t,z,x,y the order string will contain a '-' in that location. The result string will be of the same length as the number of axes. This makes it easy to loop over the dimensions.

if ids == 1 those axes will be represented in the order string as (id) where id is that axis' id. The result will be suitable for passing to order2index to get the corresponding axes, and to orderparse for dividing up into components.

`getRegion(self, *specs, **keys)`

`getRegion`

Read a region of data. A region is an n-dimensional rectangular region specified in coordinate space.

'slices' is an argument list, each item of which has one of the following forms:

- x, where x is a scalar
 - Map the scalar to the index of the closest coordinate value.
- (x,y)
 - Map the half-open coordinate interval [x,y) to index interval [x,y).
- (x,y,'cc')
 - Map the closed interval [x,y] to index interval. Other options are 'oc' (open on the left), and 'co' (open on the right, the default).
- (x,y,'co',cycle)
 - Map the coordinate interval with wraparound. If no cycle is specified, an error will occur iff axis.isCircular() is true.
 - NOTE: Only one dimension may be wrapped.
- Ellipsis
 - Represents the full range of all dimensions bracketed by None.
- ':' or None
 - Represents the full range of one dimension.

For example, suppose the variable domain is (time,level,lat,lon).

`getRegion`((10,20),850,Ellipsis,(-180,180))

retrieves:

- all times t such that 10.≤t<20.
- level 850
- all values of all dimensions between level and lon (nameless)
- longitudes x such that -180≤x≤180. This will be wrapped if lon.topology=='linear'

getSlice(self, *specs, **keys)

- x.getSlice takes arguments of the following forms and produces a return array. The keyword argument squeeze determines whether or not the shape of the returned array contains dimensions whose length is 1; by default this argument is 1, and such dimensions are 'squeezed out'.

There can be zero or more positional arguments, each of the following forms:

- a single integer n, meaning slice(n, n+1)
- an instance of the slice class
- a tuple, which will be used as arguments to create a slice object
- None or ':', which means a slice covering that entire dimension
- Ellipsis (...), which means to fill the slice list with 'Ellipsis' leaving only enough room at the end for the remaining positional arguments

There can be keyword arguments of the form key = value, where key can be one of the names 'time', 'level', 'latitude', or 'longitude'. The corresponding value can be any of (a)-(d) above.

There must be no conflict between the positional arguments and the keywords.

In (a)-(c) negative numbers are treated as offsets from the end of that dimension, as in normal Python indexing.

getTime(self)

Get the first time dimension, or None if not found

pressureRegrid(self, newLevel, missing=None, order=None, method='log')

- Return the variable regridded to new pressure levels.
- The variable should be a function of lat, lon, pressure, and <newLevel> is an axis of the result pressure levels.
- <method> is optional, either "log" to interpolate in the log space or "linear" for linear interpolation.
- <missing> and <order> are as for regrid.PressureRegridder.

rank(self)

reg_specs2slices(self, initSpecList, force=None)

regrid(self, togrid, missing=None, order=None, mask=None)

- return self regridded to the new grid. Keyword arguments are as for regrid.Regridder.

reorder(self, order)

- return self reordered per the specification order

select = __call__(self, *args, **kwargs)

- Selection of a subregion using selectors

setGrid(self, grid)

- # Set the variable grid

***specs2slices*(self, speclist, force=None)**

Create an equivalent list of slices from an index specification.
An index specification is a list of acceptable items, which are:
-- an integer
-- a slice instance (slice(start, stop, stride))
-- the object "unspecified"
-- the object None
-- a colon
The size of the speclist must be rank()

***subRegion*(self, *specs, **keys)**

Methods inherited from cdms.cdmsobj.CdmsObj:

***dump*(self, path=None, format=1)**

dump(self, path=None, format=1)

Dump an XML representation of this object to a file.
'path' is the result file name, None for standard output.
'format'==1 iff the file is formatted with newlines for readability.

***matchPattern*(self, pattern, attribute, tag)**

Match a pattern in a string-valued attribute. If attribute is None,
search all string attributes. If tag is not None, it must match tag.

***matchone*(self, pattern, attrname)**

Return true iff the attribute with name attrname is a string
attribute which matches the compiled regular expression pattern.
if attrname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is not a string.

***searchPattern*(self, pattern, attribute, tag)**

Search for a pattern in a string-valued attribute. If attribute is None,
search all string attributes. If tag is not None, it must match tag.

***searchPredicate*(self, predicate, tag)**

Apply a truth-valued predicate. Return a list containing a string
if the predicate is true and either tag is None or matches tag.
If the predicate returns false, return an empty list.

***searchone*(self, pattern, attrname)**

Return true iff the attribute with name attrname is a string
attribute which contains the compiled regular expression pattern.
if attrname is None and pattern matches at least one string
attribute. Return false if the attribute is not found or is not a string.

Methods inherited from cdms.internattr.InternalAttributesClass:

***is_internal_attribute*(self, name)**

is internal attribute(name) is true if name is internal.

`replace_external_attributes`(self, newAttributes)
 `replace_external_attributes`(newAttributes)
 Replace the external attributes with dictionary newAttributes

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

`__delattr__`(self, name)

`__getattr__`(self, name)

`__setattr__`(self, name, value)

`get_property_d`(self, name)
 Return the 'del' property handler for name that self uses.
 Returns None if no handler.

`get_property_g`(self, name)
 Return the 'get' property handler for name that self uses.
 Returns None if no handler.

`get_property_s`(self, name)
 Return the 'set' property handler for name that self uses.
 Returns None if no handler.

`set_property`(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
 Set attribute handlers for name to methods actg, acts, actd
 None means no change for that action.
 nowrite = 1 prevents setting this attribute.
 nowrite defaults to 0.
 nodelete = 1 prevents deleting this attribute.
 nodelete defaults to 1 unless actd given.
 if nowrite and nodelete is None: nodelete = 1

Methods inherited from [cdms.slabinterface.Slab](#):

`createattribute`(self, name, value)
 Create an attribute and set its name to value.

`deleteattribute`(self, name)
 Delete the named attribute.

`getattribute`(self, name)
 Get the attribute name.

`getdimattribute`(self, dim, field)
 Get the attribute named field from the dim'th dimension.
 For bounds returns the old cu one-dimensional version.

`listattributes`(self)
 Return a list of attribute names.

listdimattributes(self, dim)

 List the legal axis field names.

listdimnames(self)

 Return a list of the names of the dimensions.

setattribute(self, name, value)

 Set the attribute name to value.

showdim(self)

 Show the dimension attributes and values.

Data and other attributes inherited from [cdms.slabinterface.Slab](#):

std_slab_atts = ['filename', 'missing_value', 'comments', 'grid_name', 'grid_type', 'time_statistic', 'lo'

Methods inherited from [MA.MA.MaskedArray](#):

__and__(self, other)

 Return bitwise_and

__float__(self)

 Convert self to float.

__floordiv__(self, other)

 Return divide(self, other)

__int__(self)

 Convert self to int.

__mod__(self, other)

 Return remainder(self, other)

__or__(self, other)

 Return bitwise_or

__pos__(self)

 Return array(self)

__rand__ = ***__and__***(self, other)

 Return bitwise_and

__rfloordiv__(self, other)

 Return divide(other, self)

__rmod__(self, other)

 Return remainder(other, self)

__ror__ = ***__or__***(self, other)

 Return bitwise_or

```
__rtruediv__(self, other)
    Return divide(other, self)

__rxor__ = __xor__(self, other)
    Return bitwise_xor

__setitem__(self, index, value)
    Set item described by index. If value is masked, mask those locations.

__setslice__(self, i, j, value)
    Set slice i:j; if value is masked, mask those locations.

__truediv__(self, other)
    Return divide(self, other)

__xor__(self, other)
    Return bitwise_xor

byte_swapped(self)
    Returns the raw data field, byte_swapped. Included for consistency with Numeric but doesn't make sense in this context.

compressed(self)
    A 1-D array of all the non-masked data.

dot(self, other)
    s.dot(other) = innerproduct(s, other)

fill_value(self)
    Get the current fill value.

filled(self, fill_value=None)
    A Numeric array with masked values filled. If fill_value is None, use fill_value().  
  
If mask is None, copy data only if not contiguous.  
Result is always a contiguous, Numeric array.

ids(self)
    Return the ids of the data and mask areas

iscontiguous(self)
    Is the data contiguous?

itemsize(self)
    Item size of each data item.

mask(self)
    Return the data mask, or None. Result contiguous.

outer(self, other)
    s.outer(other) = outerproduct(s, other)
```

`put(self, values)`
Set the non-masked entries of `self` to filled(`values`).
No change to mask

`putmask(self, values)`
Set the masked entries of `self` to filled(`values`).
Mask changed to None.

`raw_data(self)`
The raw data; portions may be meaningless.
May be noncontiguous. Expert use only.

`raw_mask(self)`
The raw mask; portions may be meaningless.
May be noncontiguous. Expert use only.

`savespace(self, value)`
Set the spacesaver attribute to `value`

`spacesaver(self)`
spacesaver() queries the spacesaver attribute.

`tolist(self, fill_value=None)`
Convert to list

`toString(self, fill_value=None)`
Convert to string

`unmask(self)`
Replace the mask by None if possible.

`unshare_mask(self)`
If currently sharing mask, make a copy.

Properties inherited from MA.MA.MaskedArray:

`flat`
Access array in flat form.
`get">get = _get_flat(self)`
Calculate the flat value.
`set">set = _set_flat(self, value)`
`x.flat = value`

`imag`
Access the imaginary part of the array
`get">get = _get_imaginary(self)`
Get the imaginary part of a complex array.
`set">set = _set_imaginary(self, value)`
`x.imaginary = value`

`imaginary`
Access the imaginary part of the array

```

get">get = _get_imaginary(self)
    Get the imaginary part of a complex array.
set">set = _set_imaginary(self, value)
    x.imaginary = value

real
Access the real part of the array
get">get = _get_real(self)
    Get the real part of a complex array.
set">set = _set_real(self, value)
    x.real = value

shape
tuple giving the shape of the array
get">get = _get_shape(self)
    Return the current shape.
set">set = _set_shape(self, newshape)
    Set the array's shape.

```

Data and other attributes inherited from [MA.MA.MaskedArray](#):

```

__dict__ = <dictproxy object>
    dictionary for instance variables (if defined)

__weakref__ = <attribute '__weakref__' of 'MaskedArray' objects>
    list of weak references to the object (if defined)

handler_cache_key = 'MA.MaskedArray'

```

Functions

```

createCoordinateAxis(data, bounds=None, id=None, copy=0)
    # Create a transient axis

```

Data

```

InvalidGridElement = 'Grid domain elements are not yet implemented: '
MethodNotImplemented = 'Method not yet implemented'
NoSuchAxisOrGrid = 'No such axis or grid: '
calendarToTag = {17: '360_day', 4113: 'noleap', 4369: 'proleptic_gregorian', 69905: 'julian',
135441: 'gregorian'}
latitude_aliases = []
level_aliases = ['plev']
longitude_aliases = []
std_axis_attributes = ['name', 'units', 'length', 'values', 'bounds']
tagToCalendar = {'360': 17, '360_day': 17, '365_day': 4113, 'gregorian': 135441, 'julian': 69905,
'noleap': 4113, 'proleptic_gregorian': 4369, 'standard': 4369}
time_aliases = []

```



unspecified = 'No value specified.'